<https://leetcode.com/problems/largest-positive-integer-that-exists-with-its-negative/description/>

Given an integer array nums that **does not contain** any zeros, find **the largest positive** integer k such that -k also exists in the array.

Return **the positive integer**k. If there is no such integer, return -1.

**Example 1:**

**Input:** nums = [-1,2,-3,3]

**Output:** 3

**Explanation:** 3 is the only valid k we can find in the array.

**Example 2:**

**Input:** nums = [-1,10,6,7,-7,1]

**Output:** 7

**Explanation:** Both 1 and 7 have their corresponding negative values in the array. 7 has a larger value.

**Example 3:**

**Input:** nums = [-10,8,6,7,-2,-3]

**Output:** -1

**Explanation:** There is no a single valid k, we return -1.

**Constraints:**

1 <= nums.length <= 1000

-1000 <= nums[i] <= 1000

nums[i] != 0

**Attempt 1: 2024-01-27**

**Solution 1: Hash Table (30 min)**

**Wrong Solution (260/337)**

**Test out by: [-9,-43,24,-23,-16,-30,-38,-30], expect = -1, output = 30**

class Solution {

    public int findMaxK(int[] nums) {

        int max = -1;

        Set<Integer> set = new HashSet<>();

        for(int num : nums) {

            if(num < 0) {

                num = -num;

            }

            if(!set.add(num)) {

                if(max < num) {

                    max = Math.max(max, num);

                }

            }

        }

        return max;

    }

}

**Correct Solution**

**Style 1: HashSet**

class Solution {

    public int findMaxK(int[] nums) {

        Set<Integer> set = new HashSet<>();

        for(int num : nums) {

            set.add(num);

        }

        int max = -1;

        for(int num : set) {

            // Doesn't matter if the -num is < 0 or > 0,

            // since we will keep compare every number,

            // even its a < 0 one will definitely be

            // overwrite by a > 0 one result when set 'max'

            if(set.contains(-num)) {

                max = Math.max(max, num);

            }

        }

        return max;

    }

}

Time Complexity: O(N)

Space Complexity: O(N)

**Style 2: Frequency Array instead of HashSet**

class Solution {

    public int findMaxK(int[] nums) {

        // Because -1000 <= nums[i] <= 1000, in freq array

        // we have to shift position i right for 1000 to

        // guarantee 0-based index

        int[] freq = new int[2001];

        for(int num : nums) {

            freq[num + 1000]++;

        }

        for(int i = 0; i < 2001; i++) {

            // If still 'i < 1000' means before shift index right

            // for 1000, its a negative value(i - 1000 < 1000 - 1000

            // -> i - 1000 < 0), the logic behind to find largest

            // positive integer, we would like to find its mapping

            // smallest negative integer in this 2001 buckets array

            // first, and we have 'freq[i]' represents '-k' and

            // 'freq[2000 - k]' represents 'k', in for loop if we

            // start encountering a 'freq[i] > 0', we may find a

            // candidate for smallest negative value, we mention its

            // 'candidate' because if its corresponding 'k' representative

            // 'freq[2000 - i] > 0' not exist, this candidate 'freq[i]'

            // won't stand, only when its mapping positive value

            // 'freq[2000 - i]' also exist, we find a pair, and

            // the largest positive integer k will be '1000 - i',

            // otherwise we continue for loop to find next candidator

            if(i < 1000 && freq[i] > 0 && freq[2000 - i] > 0) {

                return 1000 - i;

            }

        }

        return -1;

    }

}

Time Complexity: O(N)

Space Complexity: O(N)